M2351 USB Component Firmware Update

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code demonstrates the implementation of Component Firmware Update (CFU) functionality on the M2351 development platform.
BSP Version	M2351_Series_BSP_CMSIS_V3.00.006
Hardware	NuMaker-PFM-M2351 V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

This example code demonstrates a USB device that supports component firmware update with the NuMaker-PFM-M2351 evaluation board. When the PC installed with the update information from Windows Update connects with the USB device, the CFU progress will start to update the sub-component of the USB device.

1.1 NuMaker-PFM-M2351 Evaluation Board

The NuMaker-PFM-M2351 is an evaluation board based on the Nuvoton NuMicro® M2351 microcontroller. The M2351 secure microcontroller series is empowered by Arm® Cortex®-M23 core with TrustZone® for Armv8-M architecture, which elevates the traditional firmware security to the new level of robust software security.

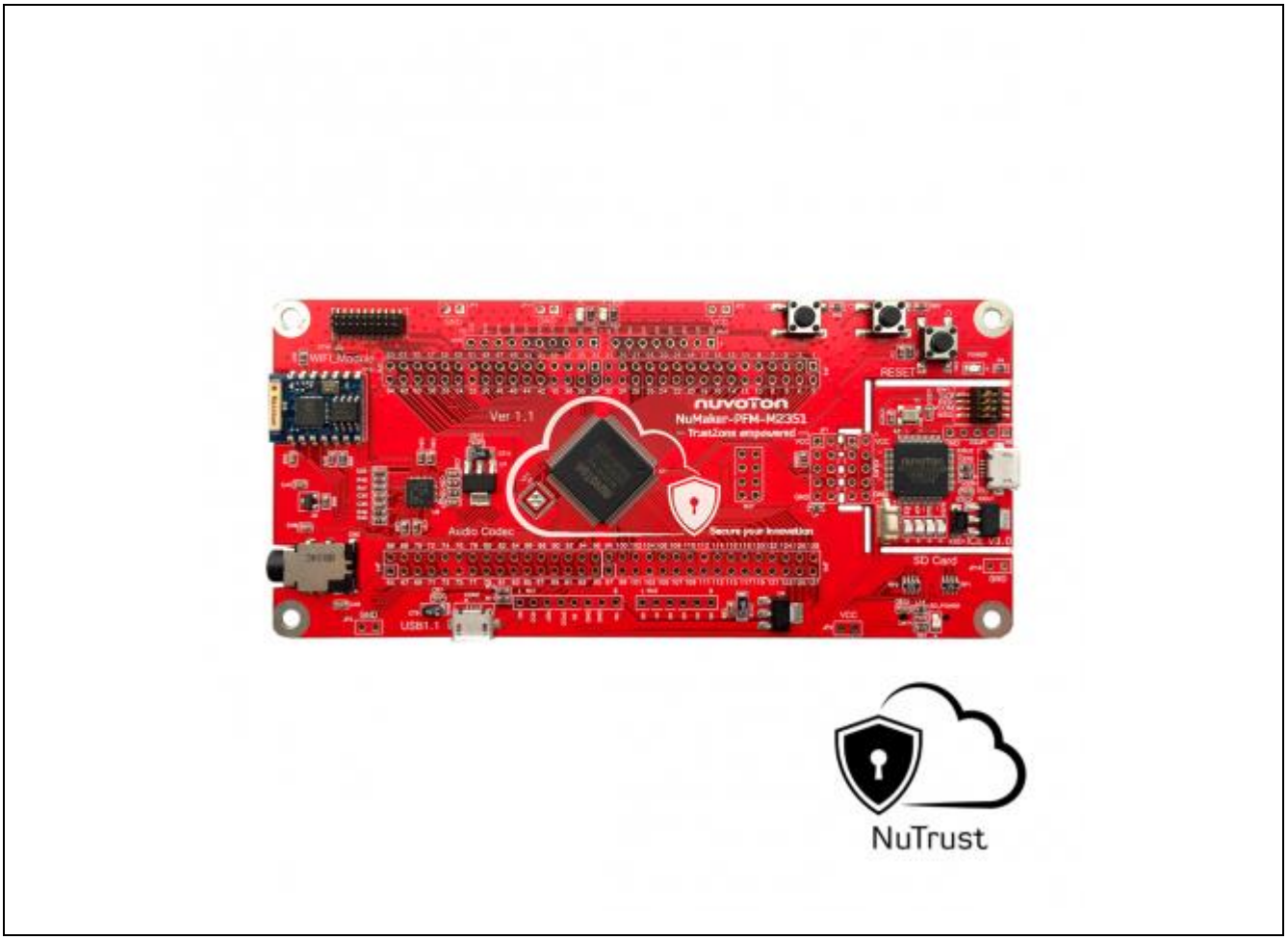


Figure 1-1 NuMaker-PFM-M2351 Evaluation Board

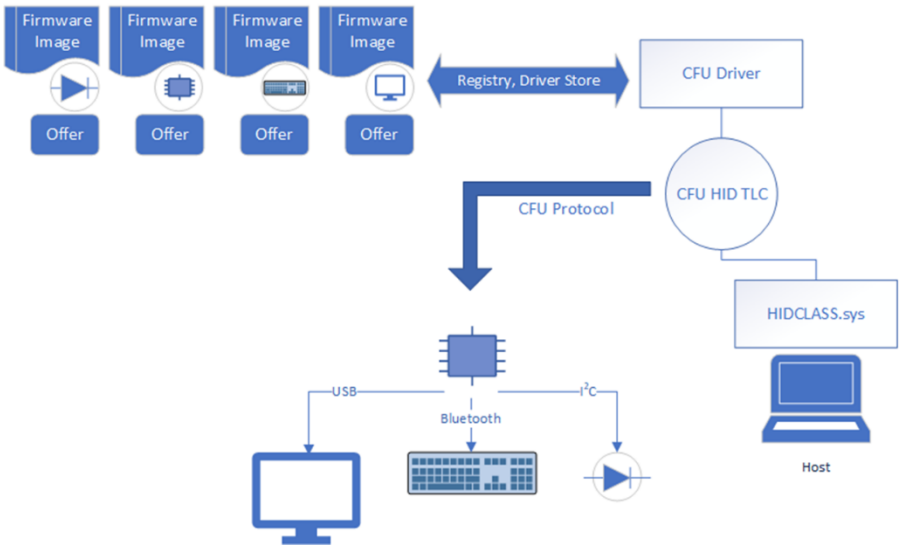
1.2 Analysis of CFU Working Principles

1.2.1 Background Introduction

The Component Firmware Update (CFU) is a service provided by Microsoft for automatically updating hardware firmware to enhance device performance and security. CFU occurs with little or no user disruption – no “update mode” that requires the user to wait or even be aware that an update is taking place. Also, CFU is basically automatic updated: The system detects and installs the latest firmware updates automatically without user intervention.

An CFU is delivered through Windows Update drivers, and the target must be able to reject firmware before it is downloaded if it is inappropriate.

1.2.2 Diagram of Before Starting CFU Progress



https://blogs.windows.com/windowsdeveloper/2018/10/17/introducing-component-firmware-update/

Figure 1-2 Architecture Diagram

1. Firmware providers upload their driver by using Windows Update.
2. Users receive the driver at the host side.
3. When the target that fits the driver setting is connected with host, the host will start the CFU progress.
4. The firmware of the sub-components will be updated during the CFU progress by receiving the data from the main target component.

1.2.3 Details of Transfer Data Format

CFU submissions to the resident firmware are file pairs: one file is the offer part, and the other file is the content part. A CFU driver (host) may contain multiple firmware images for a primary component and sub-components associated with the component.

The offer contains enough information about the payload to allow the primary component to decide if it is an acceptable payload.

Offer information includes a CFU protocol version, component ID (and sub-component ID if applicable), firmware version, release vs. debug status, and other information.

The payload of a package is basically a set of addresses and fixed-size arrays of bytes to be programmed. The bytes are opaque to the host.

1.2.4 Details of CFU Process

The general firmware update process by using CFU is for the host to issue the offer of each package to the primary component.

When the host detects the device that matches the firmware is connected, it sends a host initialized notification to the device, telling the device the CFU update process is ready to go and waiting for the receiving. Then the host sends an offer start notification.

After that, the host will send the component offers in sequence. For each independent component offer, primary component can accept, reject, or skip the offer. If the offer is accepted, the host will start to send the data for update and start the update progress, else it will pass and send another component's offer in order.

This will be ended until the device always returns reject while receive every component offer in an offer-chain. Then the host sends an offer end notification and finishes the CFU progress.

1.3 Analysis of the USB Descriptors

1.3.1 Overview of Component USB Descriptors

USB descriptors are a set of structured data used to describe the attributes, functionalities, and requirements of a USB device. These descriptors help the host identify the device type, available features, and transmission needs.

1.3.2 Role of Descriptors in Component Device

In a component device that handles the CFU process, descriptors provide detailed information on how the device interacts with the host and transmits data. The following outlines the specific role of each descriptor in video transmission and its implementation in the code.

1.3.2.1 Device Descriptor

The device descriptor provides basic information about the USB device, such as device type, vendor ID, product ID, and the supported USB version. This is the first step for the host to obtain device information, allowing the host to identify the basic attributes of the device.

```
static uint8_t s_au8DeviceDescriptor [] = {
    LEN_DEVICE,     /* bLength */
    DESC_DEVICE,    /* bDescriptorType */
#ifdef SUPPORT_LPM
    0x01, 0x02,     /* bcdUSB >= 0x0201 to support LPM */
#else
    0x10, 0x01,     /* bcdUSB */
#endif
    0x00,           /* bDeviceClass */
    0x00,           /* bDeviceSubClass */
```

```
    0x00,           /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_CFU_VID & 0x00FF,
    (USBD_CFU_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_CFU_PID & 0x00FF,
    (USBD_CFU_PID & 0xFF00) >> 8,
    0x00, 0x00,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x00,           /* iSerialNumber - no serial */
    0x01            /* bNumConfigurations */
};
```

1.3.2.2 Configuration Descriptor

The configuration descriptor describes the device's configuration options, including power usage and the number of supported interfaces. The host uses this descriptor to configure the device so that it can function properly. It also contains the HID descriptor, interface descriptor and Endpoint descriptor.

The interface descriptor provides detailed information about each interface in the configuration, including the functions they provide (such as video control or video streaming) and the number of endpoints used.

The endpoint descriptor defines the specific method of data transmission between the host and the device. For video streaming, endpoints are typically configured for isochronous transfer to ensure data stability and real-time transmission. In this example code, Endpoint 2 and Endpoint 3 is being used, so their information must be on the endpoint descriptor.

Code Implementation:

```
cstatic uint8_t s_au8ConfigDescriptor[] =
{
    LEN_CONFIG,     /* bLength */
    DESC_CONFIG,    /* bDescriptorType */
    /* wTotalLength */
    CONFIG_DESC_TOTAL_LENGTH & 0x00FF,
    (CONFIG_DESC_TOTAL_LENGTH & 0xFF00) >> 8,
    0x01,           /* bNumInterfaces */
    0x01,           /* bConfigurationValue */
    0x00,           /* iConfiguration */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),/* bmAttributes */
    USBD_MAX_POWER,         /* MaxPower */

    /* I/F descr: HID */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x00,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x02,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
```

```
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

    /* HID Descriptor */
    LEN_HID,                            /* Size of this descriptor in UINT8s. */
    DESC_HID,                           /* HID descriptor type. */
    0x10, 0x01,                         /* HID Class Spec. release number. */
    0x00,                               /* H/W target country. */
    0x01,                               /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,                       /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(g_CfuVirtualHid_HidReportDescriptor) & 0x00FF,
    (sizeof(g_CfuVirtualHid_HidReportDescriptor) & 0xFF00) >> 8,
    /* EP Descriptor: interrupt in. */
    LEN_ENDPOINT,                       /* bLength */
    DESC_ENDPOINT,                      /* bDescriptorType */
    (CFU_INT_IN_EP_NUM | EP_INPUT),     /* bEndpointAddress */
    EP_INT,                             /* bmAttributes */
    /* wMaxPacketSize */
    EP2_MAX_PKT_SIZE & 0x00FF,
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,        /* bInterval */
    /* EP Descriptor: interrupt out. */
    LEN_ENDPOINT,                       /* bLength */
    DESC_ENDPOINT,                      /* bDescriptorType */
    (CFU_INT_OUT_EP_NUM | EP_OUTPUT),   /* bEndpointAddress */
    EP_INT,                             /* bmAttributes */
    /* wMaxPacketSize */
    EP3_MAX_PKT_SIZE & 0x00FF,
    (EP3_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,        /* bInterval */
};
```

1.3.2.3 CFU HID Report Descriptor

The CFU HID report descriptor records important information that helps the host to find the target to do the CFU progress. In this descriptor, each parameter should be checked to match the driver, such as USAGE_PAGE and REPORT_ID for offer/payload input and output.

```
static uint8_t g_CfuVirtualHid_HidReportDescriptor[] =
{
    0x06, CFU_DEVICE_USAGE_PAGE,        // USAGE_PAGE(0xFA00)
    0x09, CFU_DEVICE_USAGE,             // USAGE(0xF5)
    0xA1, 0x01,                         // COLLECTION(0x01)

    0x15, 0x00,                         // LOGICAL_MINIMUM(0)
    0x27, 0xFF, 0xFF, 0xFF, 0xFF,       // LOGICAL_MAXIMUM(-1)

    // Necessary to work around possible bug in HID class related to VHF.

    0x85, REPORT_ID_PAYLOAD_INPUT,      // REPORT_ID(34)
    0x75, INPUT_REPORT_LENGTH,          // REPORT SIZE(32)
    0x95, 0x04,                         // REPORT COUNT(4)
    0x19, PAYLOAD_INPUT_USAGE_MIN,      // USAGE MIN (0x26)
    0x29, PAYLOAD_INPUT_USAGE_MAX,      // USAGE MAX (0x29)
```

```
    0x81, 0x02,                         // INPUT(0x02)

    0x85, REPORT_ID_OFFER_INPUT,        // REPORT_ID(37)
    0x75, INPUT_REPORT_LENGTH,          // REPORT SIZE(32)
    0x95, 0x04,                         // REPORT COUNT(4)
    0x19, OFFER_INPUT_USAGE_MIN,        // USAGE MIN (0x1A)
    0x29, OFFER_INPUT_USAGE_MAX,        // USAGE MAX (0x1D)
    0x81, 0x02,                         // INPUT(0x02)

    0x85, REPORT_ID_PAYLOAD_OUTPUT,     // REPORT_ID(33)
    0x75, 0x08,                         // REPORT SIZE(8)
    0x95, OUTPUT_REPORT_LENGTH,         // REPORT COUNT(60)
    0x09, PAYLOAD_OUTPUT_USAGE,         // USAGE(0x31)
    0x92, 0x02, 0x01,                   // OUTPUT(0x02)

    0x85, REPORT_ID_OFFER_OUTPUT,       // REPORT_ID(36)
    0x75, INPUT_REPORT_LENGTH,          // REPORT SIZE(32)
    0x95, 0x04,                         // REPORT COUNT(4)
    0x19, OFFER_OUTPUT_USAGE_MIN,       // USAGE MIN (0x1E)
    0x29, OFFER_OUTPUT_USAGE_MAX,       // USAGE MAX (0x21)
    0x91, 0x02,                         // OUTPUT(0x02)

    0x85, REPORT_ID_VERSIONS_FEATURE,   // REPORT_ID(32)
    0x75, 0x08,                         // REPORT SIZE(8)
    0x95, FEATURE_REPORT_LENGTH,        // REPORT COUNT(60)
    0x09, VERSIONS_FEATURE_USAGE,       // USAGE(0x42)
    0xB2, 0x02, 0x01,                   // FEATURE(0x02)

    0xC0                                // END_COLLECTION()
};
```

1.4 Driver Information

In the CFU process, after the firmware and driver pass the necessary test, firmware provider can upload their driver by using Window Update. Host machine can easily get the driver during regular Window Update.

In test stage, providers should create, package and test their driver in independent environments. This can be implemented by using the sample project provided from Microsoft.

It is recommended to build the driver by using the project that Microsoft provided at GitHub: microsoft/CFU: Component Firmware Update (github.com), which contains several types of the examples in the folder Host.

For the implementation and details, please refer to the following link provided from Microsoft: Component Firmware Update (CFU) INF Configuration - Windows drivers | Microsoft Learn

The folder *VirtualDevicePackage* provides an example driver for test. Although host machine should get the driver and let the driver automatically be installed from Windows Update during CFU progress, developers can install and test the driver manually by using *.inf* file.

1.5 Example Driver installation

The Example Driver is different from the regular driver used in CFU progress since it does not go through the Windows Update path. To test that the example driver can be installed and work in the user environment, the user needs to follow the steps below.

Before installing the driver, the user needs to register the digital signature. Double-click the *.cer* file named virtualdevicefirmwareupdate.cer and click the “Install Certification …” button. The Certificate Import Wizard window should appear. Choose the Store Location as “Local Machine” and click the Next button with administrator permission. Then, on the Certificate Store page, choose the “Place all certificates in the following store” option, click Browse, and select “Trusted Root Certification Authorities” to store the certificate.

After the certificate is registered, right-click the *virtualdevicefirmwareupdate.inf* file and choose the install option; the installation of the *.inf* driver will then be completed.

1.6 Demo Result

First, make sure that the driver has been installed at the host machine. When the main component is connected to the host machine by USB, check the firmware part of the device manager. If the driver setting fit with the device, but the name of the driver is not shown or not same as the driver setting, then try to manually update driver.

To update the driver manually, user needs to find the Firmware that has the same hardware ID set in the *.inf* file at the Device Manager, HID\VID_0416&UP:FA00_U:00F5 in the example case. Then right-click and choose the update driver option and choose the “NuvotonDevice Firmware Update” driver and update.

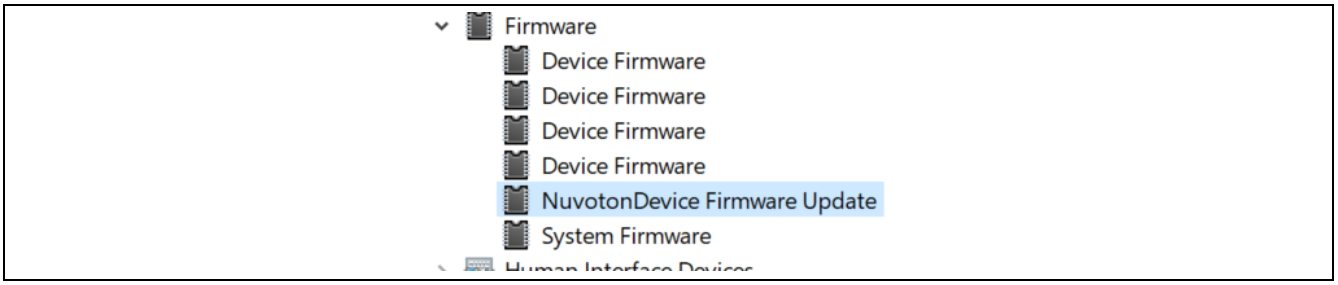


Figure 1-3 Device Manager

When the board with proper configurations has connected to the host, the CFU process will automatically start. By connecting the host with ICE to the NuMaker-PFM-M2351 board, VCOM can be used to check the output, which can show the transfer information between host and primary component.

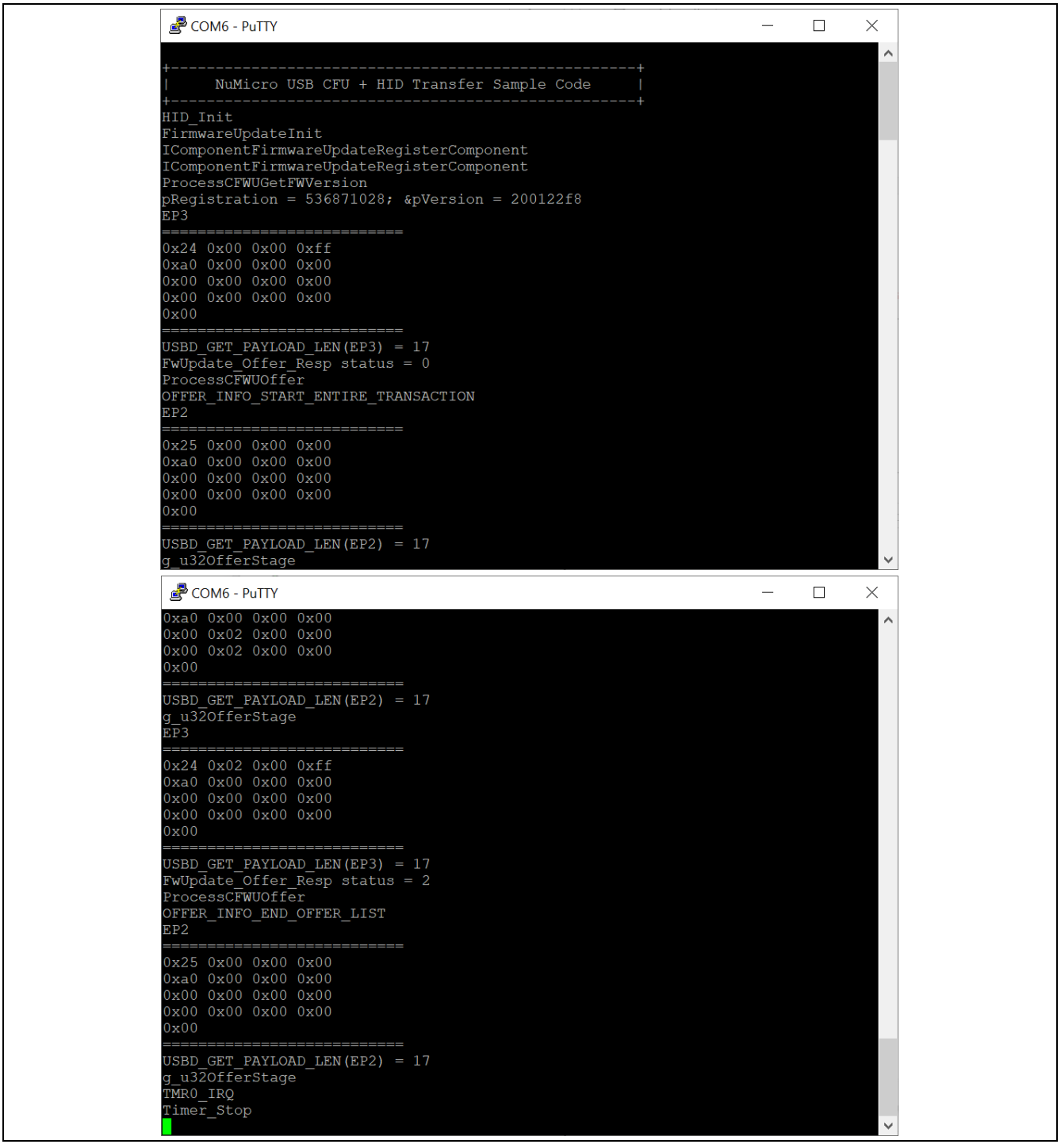


Figure 1-4 UART Message

Since the transfer between primary component and sub-component will be implemented by the user independently, these messages will only concern about the correctness of the CFU progress. Also, it is important to set the timeout; otherwise, it may fall into endless offer if the implementation of the offer accept/skip/reject goes wrong.

2. Code Description

The main program of the component device example code is as follows: after initializing the system, main program sets up the USB Device and waits for CFU offer that is sent from host machine by using the USB interrupt.

```
int32_t main(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();
    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();
    /* Init UART for printf */
    UART0_Init();
    printf("\n");
    printf("+-----------------------------------------------------+\n");
    printf("|     NuMicro USB CFU + HID Transfer Sample Code      |\n");
    printf("+-----------------------------------------------------+\n");
    USBD_Open(&gsInfo, HID_ClassRequest, NULL);
    /* Endpoint configuration */
    HID_Init();
    USBD_Start();
    NVIC_EnableIRQ(USBD_IRQn);
#if CRYSTAL_LESS
    /* Backup default trim */
    s_u32DefaultTrim = M32(TRIM_INIT);
    s_u32LastTrim = s_u32DefaultTrim;
#endif
    /* Clear SOF */
    USBD->INTSTS = USBD_INTSTS_SOFIF_Msk;
    /* Enable FMC ISP function */
    FMC_Open();
    FirmwareUpdateInit();
    // Register the component chain
    IComponentFirmwareUpdateRegisterComponent(&s_Comp_Registration);
    IComponentFirmwareUpdateRegisterComponent(&s_Comp_Registration_2);
    ProcessCFWUGetFWVersion(&s_Get_FwVer_Resp);

    while (1)
    {
#if CRYSTAL_LESS

        /* Start USB trim if it is not enabled. */
        if ((SYS->TCTL48M & SYS_TCTL48M_FREQSEL_Msk) != 1)
        {
            /* Start USB trim only when SOF */
            if (USBD->INTSTS & USBD_INTSTS_SOFIF_Msk)
            {
                /* Clear SOF */
                USBD->INTSTS = USBD_INTSTS_SOFIF_Msk;
                /* Re-enable crystal-less */
                SYS->TCTL48M = 0x01;
                SYS->TCTL48M |= SYS_TCTL48M_REFCKSEL_Msk;
            }
```

```
        }

        /* Disable USB Trim when error */
        if (SYS->TISTS48M & (SYS_TISTS48M_CLKERRIF_Msk | SYS_TISTS48M_TFAILIF_Msk))
        {
            /* Last TRIM */
            M32(TRIM_INIT) = s_u32LastTrim;
            /* Disable crystal-less */
            SYS->TCTL48M = 0;
            /* Clear error flags */
            SYS->TISTS48M = SYS_TISTS48M_CLKERRIF_Msk | SYS_TISTS48M_TFAILIF_Msk;
            /* Clear SOF */
            USBD->INTSTS = USBD_INTSTS_SOFIF_Msk;
        }

        /* Check trim value whether it is over the threshold */
        if ((M32(TRIM_INIT) > (s_u32DefaultTrim + TRIM_THRESHOLD)) ||
                (M32(TRIM_INIT) < (s_u32DefaultTrim - TRIM_THRESHOLD)))
        {
            /* Write updated value */
            M32(TRIM_INIT) = s_u32LastTrim;
        }
        else
        {
            /* Backup trim value */
            s_u32LastTrim = M32(TRIM_INIT);
        }

#endif
    }
}
```

In this example code, Endpoint 2 and Endpoint 3 handle the input and output of the HID CFU communication.

```
void EP2_Handler(void)  /* Interrupt IN handler */
{
    uint8_t *pu8Ptr;
    CFU_SetInReport();
}

void EP3_Handler(void)  /* Interrupt OUT handler */
{
    uint8_t *pu8Ptr;
    printf("EP3\n");
    printf("==========================\n");
    /* Interrupt OUT */
    pu8Ptr = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));

    for (int i = 0; i < USBD_GET_PAYLOAD_LEN(EP3); i ++)
    {
        printf("0x%02x ", pu8Ptr[i]);

        if (i % 4 == 3 || i == USBD_GET_PAYLOAD_LEN(EP3) - 1) printf("\n");
    }
```

```
    printf("==========================\n");
    printf("USBD_GET_PAYLOAD_LEN(EP3) = %d\n", USBD_GET_PAYLOAD_LEN(EP3));
    CFU_GetOutReport(pu8Ptr + 1, USBD_GET_PAYLOAD_LEN(EP3) - 1);
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
}
```

In order for the program to continue with the progress when receiving the CFU data, each component should register its information on the register chain by using COMPONENT_REGISTRATION structure, which contains the component ID, necessary functions' pointers, and the pointer of the next registration.

```
typedef struct COMPONENT_REGISTRATION_STRUCT
{
    struct COMPONENT_REGISTRATION_STRUCT *pNext;
    const ICOMPONENT_INTERFACE interface;
    const UINT8 componentId;
} COMPONENT_REGISTRATION;

typedef struct
{
    MCU_STATUS(*GetVersion)(UINT32* pVersion);
    MCU_STATUS(*GetProductInfo)(UINT32* pProductInfo);
    MCU_STATUS(*ProcessOffer)(FWUPDATE_OFFER_COMMAND* pCommand,
                              FWUPDATE_OFFER_RESPONSE* pResponse);
    MCU_STATUS(*GetCrcOffset)(UINT32* pOffset);
    MCU_STATUS(*NotifySuccess)(BOOL forceReset,
                               READ_FIRMWARE_FUNC readHandler,
                               READ_COMPLETED_FUNC readCompleteHandler);
} ICOMPONENT_INTERFACE;
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP Version
  - M2351_Series_BSP_CMSIS_V3.00.006
- Driver
  - Sample driver in folder */VirtualDevicePackage* or own driver

3.2 Hardware Requirements

- Circuit components
  - NuMaker-PFM-M2351 V1.0
    - ♦ Connect USB to USB 1.1 for main program.
    - ♦ Connect UART to ICEV3.0 (VCOM) for message print.

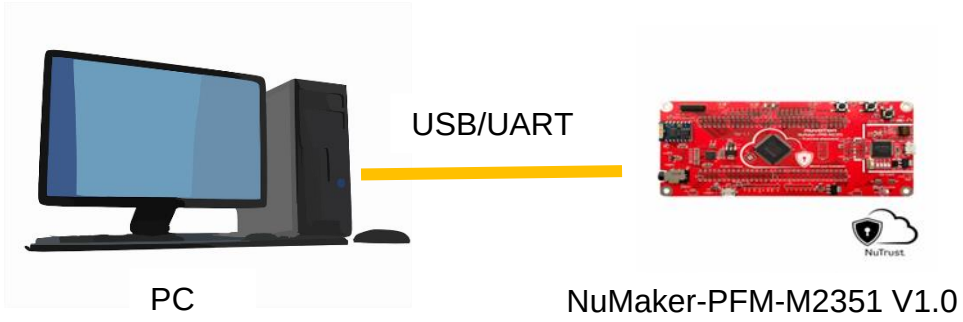


Figure 3-1 USB CFU Connection Execution Diagram

4. Directory Information

The directory structure is shown below.

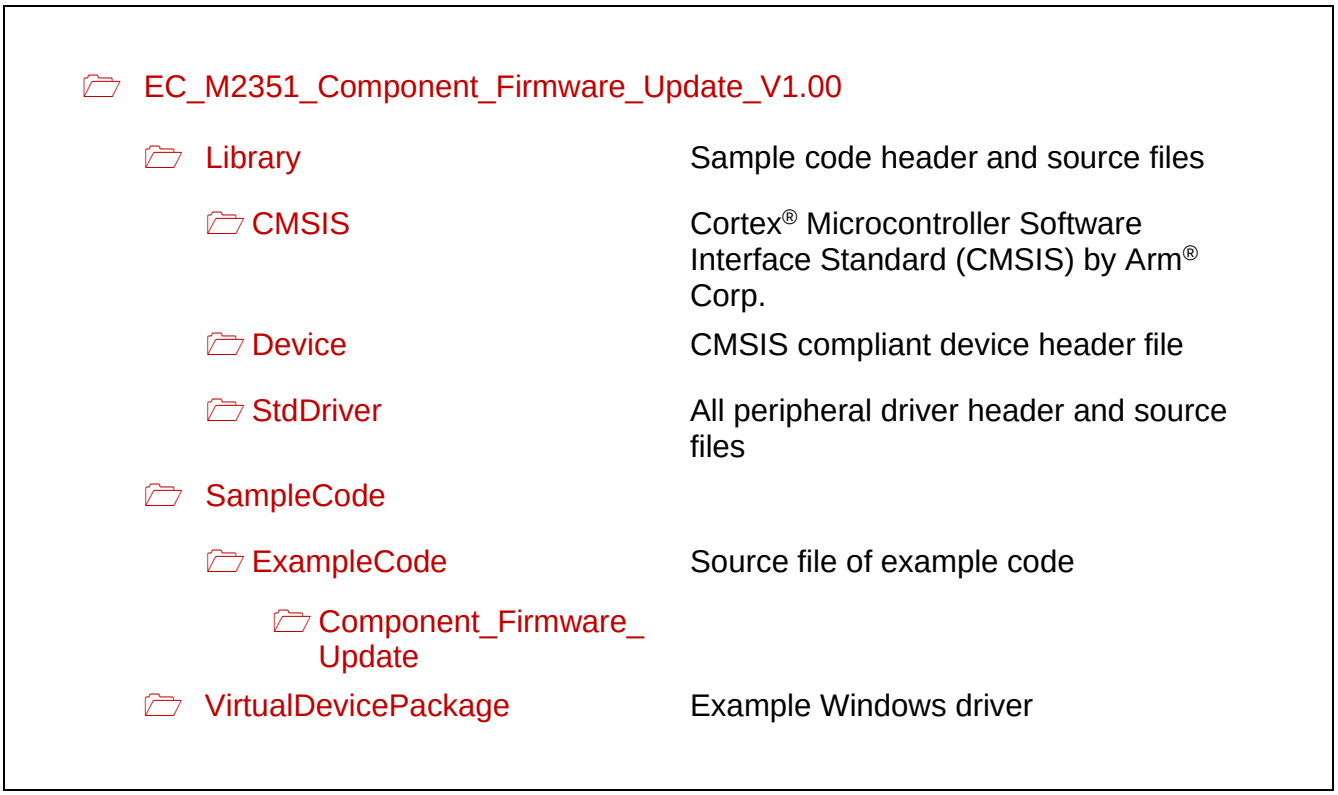


Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *Component_Firmware_Update.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run
   - Show the results by using UART interface

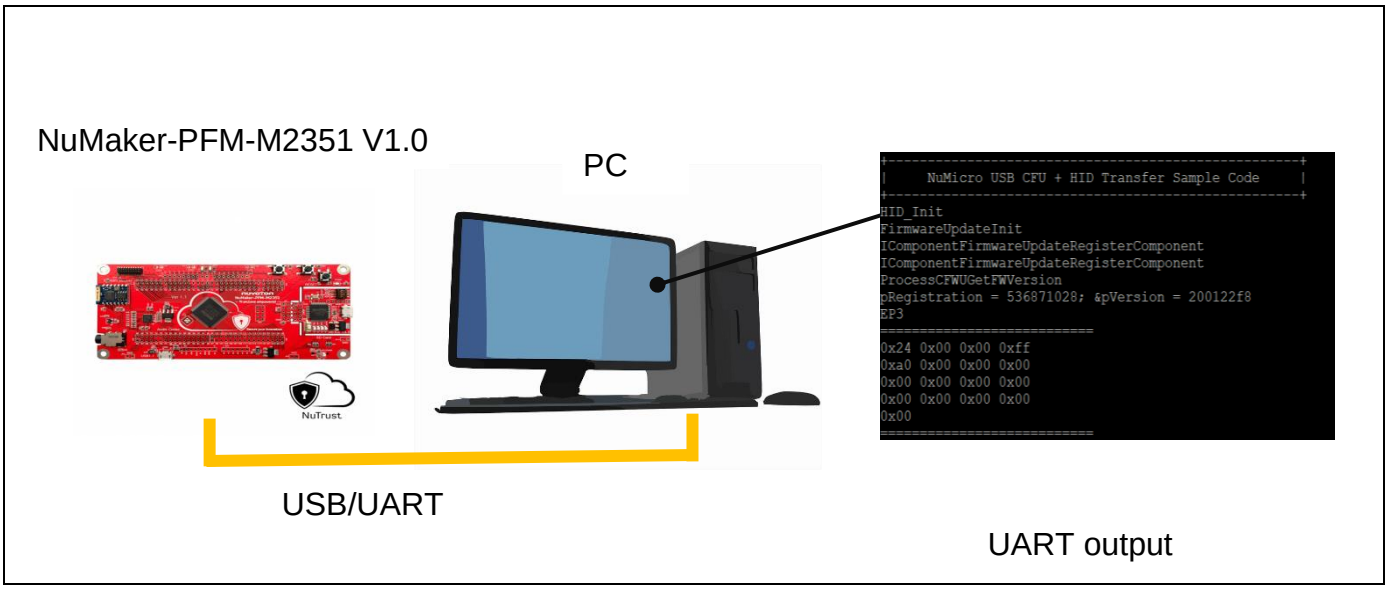


Figure 5-1 Results of the USB CFU

6. Revision History

Date	Revision	Description
2024.11.21	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.